Outline

- Programmable units, capabilities
- High level shading languages
- GPGPU (General-Purpose Computation on the GPU)

Review of GPU Pipeline

- Fixed Functionality Pipeline

Programmable Units

- Programming Graphics Hardware

High Precision

- 32-bit IEEE floating-point throughout pipeline
  - Framebuffer
  - Textures
  - Fragment processor
  - Vertex processor
  - Interpolants

Multiple data types in hardware

- Shader also supports:
  - 16-bit “half” floating point, 12-bit fixed point
  - These may be faster than 32-bit
  - 8-bit Integer

- Framebuffer/textures also support:
  - Large variety of color channel formats
    - E.g., classical 8-bit per component RGBA, BGRA, etc.
  - These formats use less memory bandwidth than FP32
Computation capabilities

- 4-vector FP32 operations (comp. to sse)
- Native vector/matrix instructions, common math functions
- Branching + true data-dependent control flow
- Powerful texture access
  - Think of it as a memory-read instruction, with optional user-controlled filtering (bilinear, mipmap)

Vertex Processing Unit

- Two tasks
  - Manipulate geometry (vertex level)
  - Prepare data (attributes) for pixel shader

- Texture access
  - Think of it as a memory-read instruction, with optional user-controlled filtering (bilinear, mipmap)

Vertex Shaders

- 50 face geometries
  - angry, happy, sad, move eyebrow,…
- Each target stored as difference vector
  - For each vertex: average position + 50 differences
- Result is a weighted sum of all targets
  - Big multiply-add
  - Per active blend target
  - Per attribute
- Skinning

Rasterization

- Non-programmable
- Generate fragments
- Interpolate attributes
  - Perspective correct interpolation?

Fragment Processing Unit

- Non-programmable
- Generate fragments
- Interpolate attributes
  - Perspective correct interpolation?
Pixel Shaders

- Each pixel is calculated individually
- Pixel shaders have limited or no knowledge of neighbouring pixels

Applications

- Shading effects
  - Per-pixel Shading, Arbitrary BRDF
  - NPR (non-photorealistic rendering)
- Image processing
  - Image filtering
  - Image warping
  - HDR tone mapping
- General-Purpose Computation

Reflections and refractions

Toon Shader

- This is a simple “toon shader” designed to give a cartoonish look to the geometry
- Customized Fragment Shader

Image processing

- Start with ordinary model
  - Render to backbuffer
- Render parts that are the sources of glow
  - Render to offscreen texture
- Blur the texture
- Add blur to the scene

Multipass Rendering

- Examples:
  - Shadow mapping
  - Planar reflections
  - Procedurally generated textures
  - …
Shading Languages

- Assembly code
- Higher-level language and compiler
  - GLSL: GLslang (OpenGL ARB)
  - HLSL: used with Dx (Microsoft)
- Programs compiled and loaded at run-time
- Compiler can do certain optimizations

Programming in assembly is not fun

<table>
<thead>
<tr>
<th>Assembly</th>
<th>GLSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRC R3.y, C11.w;</td>
<td>Lweight = (timeval - floor(timeval));</td>
</tr>
<tr>
<td>ADD R3.x, C11.y, -R2.y;</td>
<td>Lweight = 1.0 - Lweight;</td>
</tr>
<tr>
<td>MOV R4.y, R3.y;</td>
<td>normalize = floor(timeval)/64.0 +</td>
</tr>
<tr>
<td>ADD R4.x, -R4.y, C4.w;</td>
<td>1.0/128.0;</td>
</tr>
<tr>
<td>MUL R5.xy, R3.xy, C11.xyw;</td>
<td>normalize = normalize + 1.0/64.0;</td>
</tr>
<tr>
<td>ADD R5.xy, -R2.y, C11.y;</td>
<td>loffset = f2tex2D(tex2);</td>
</tr>
<tr>
<td>TEX H4, R3, TEX2, 2D;</td>
<td>floord2(coord2, 1.0/128.0);</td>
</tr>
<tr>
<td>ADD R5.x, R3.x, C11.x;</td>
<td>loffset = f2tex2D(tex2);</td>
</tr>
<tr>
<td>TEX H6, R3, TEX2, 2D;</td>
<td>floord2(coord2, 1.0/128.0);</td>
</tr>
</tbody>
</table>

... etc.

GLSL example

- OpenGL Shading Language
- Vertex shader

```c
uniform float some_constant;
varying vec3 point, normal;

void main(void)
{
  gl_Position = ftransform();
  point = gl_ModelViewMatrix * gl_Vertex;
  normal = gl_ModelViewMatrix * gl_Normal;
}
```

- Fragment Shader

```c
uniform vec3 light_pos, Id, Is, kd, ks;
varying vec4 point, normal;

void main(void)
{
  vec3 L = normalize(light_pos - point);
  vec3 N = normalize(normal);
  vec3 V = -normalize(point);
  vec3 R = reflect(V, N);
  vec3 diffuse = Id * kd * max(dot(N, L), 0.0);
  vec3 specular = Is * ks * pow(max(dot(R, V), 0.0), 20.0);
  // gamma correction
  gl_FragColor = pow(diffuse + specular, 1.0/2.2);
}
```

Shading Language Summary

- C-like language – expressive and efficient
- HW data types
- Vector and matrix operations
- Write separate vertex and fragment programs
- Connectors enable mix & match of data flows

GPGPU

- General Purpose computation on GPU
  - GPU has enormous computational power
  - Vector machine, stream processing model
  - It is directly connected to the frame buffer
  - Suitable for applications that are computation intensive and inherently parallel (data independence)
<table>
<thead>
<tr>
<th>Applications</th>
<th>What are the challenges with GPGPU?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital image processing</td>
<td>Non-uniform data are difficult to handle</td>
</tr>
<tr>
<td>Fourier transform</td>
<td>No support for pointers, structures based on pointers</td>
</tr>
<tr>
<td>Geometric computing</td>
<td>will be non-trivial to implement</td>
</tr>
<tr>
<td>Volume rendering</td>
<td>No recursive function calls</td>
</tr>
<tr>
<td>Ray tracing</td>
<td>No bitwise operations</td>
</tr>
<tr>
<td>Database</td>
<td></td>
</tr>
</tbody>
</table>